Interface Segregation Principle (ISP)

Produced by:

Dr. Siobhán Drohan (sdrohan@wit.ie)

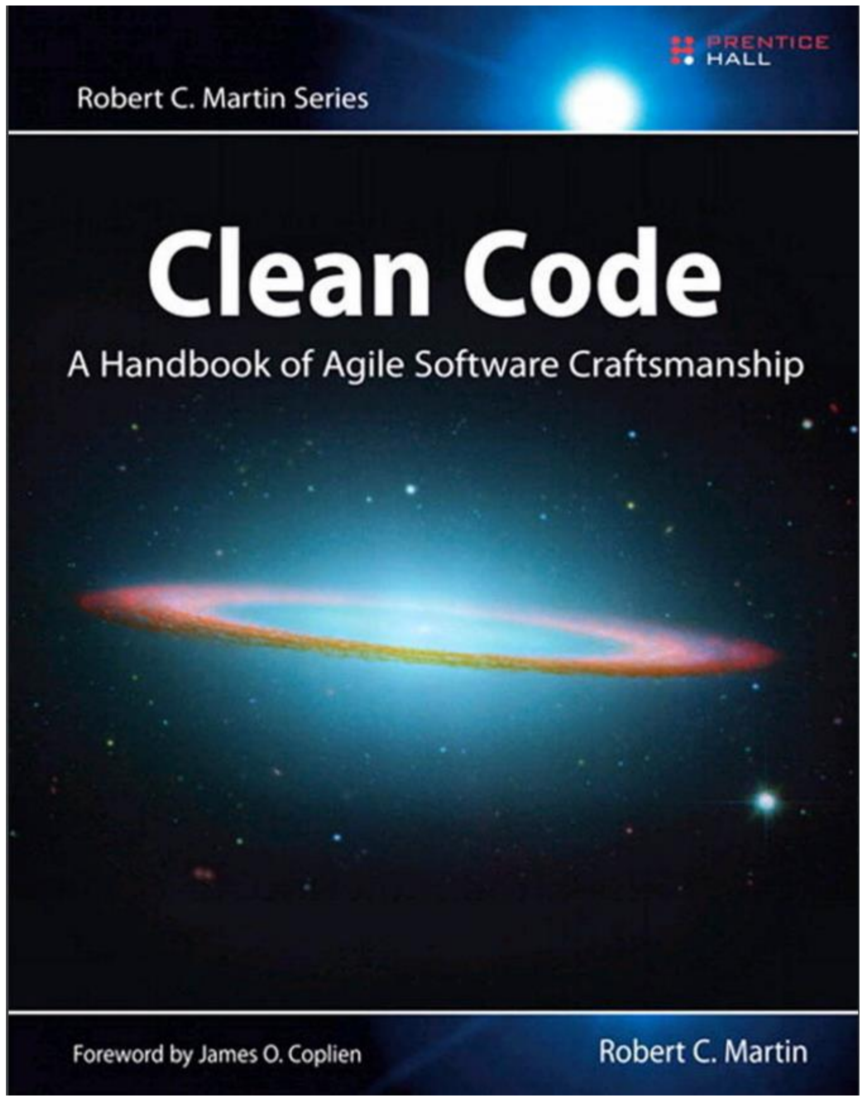
Eamonn de Leastar (edeleastar@wit.ie)

SOLID Class Design Principles

In this talk, we will refer to the SOLID principles examples in this book and also this website.

SOLID → five principles for object-oriented class design i.e.

best guidelines for building a maintainable object-oriented system.


PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

SOLID Class Design Principles

S Single Responsibility Principle (SRP). Classes should have one, and only one, reason to change. Keep your classes small and single-purposed.

O Open-Closed Principle (OCP). Design classes to be open for extension but closed for modification; you should be able to extend a class without modifying it. Minimize the need to make changes to existing classes.

L Liskov Substitution Principle (LSP). Subtypes should be substitutable for their base types. From a client's perspective, override methods shouldn't break functionality.

I **Interface Segregation Principle (ISP). Clients should not be forced to depend on methods they don't use. Split a larger interface into a number of smaller interfaces.**

D Dependency Inversion Principle (DIP). High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.

Interface Segregation Principle
If IRequireFood, I want to Eat(Food food) not,
LightCandelabra() or LayoutCutlery(CutleryLayout preferredLayout)

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

Interface Segregation Principle (ISP) – Basic Principle

"Don't depend on things you don't need"

Interface Segregation Principle
If IRequireFood, I want to Eat(Food food) not,
LightCandelabra() or LayoutCutlery(CutleryLayout preferredLayout)

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

Interface Segregation Principle (ISP) – Basic Principle

Clients should not be forced to depend upon interface members they do not use.

Instead, create multiple, smaller, cohesive interfaces.

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

Interface Segregation Principle (ISP) – Basic Principle

Clients should not be forced to depend upon interface members they do not use.

Instead, create multiple, smaller, cohesive interfaces.

Smaller interfaces are easier to implement → improves flexibility and the possibility of reuse.

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

Example 1 -

A Basic ATM

Interface Segregation Principle
If IRequireFood, I want to Eat(Food food) not,
LightCandelabra() or LayoutCutlery(CutleryLayout preferredLayout)

ISP – Example 1 – A Basic ATM

Picture an ATM machine which has a screen where we wish to display different messages.

Assume that messages can be displayed at different times during an ATM transaction e.g.

- Login
- Withdrawal
- Deposit
- etc.

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

Possible Interface Solution:

```
public interface Messenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
        void tellCardWasSiezed();
        void askForAccount();
        void tellNotEnoughMoneyInAccount();
        void tellAmountDeposited();
        void tellBalance();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

Fat Interface

Possible Interface Solution:

```
public interface Messenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
        void tellCardWasSiezed();
        void askForAccount();
        void tellNotEnoughMoneyInAccount();
        void tellAmountDeposited();
        void tellBalance();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

BUT…we have grouped unrelated functionality together…yes, they are all messages, but they occur at different times during an ATM transaction.

```
public interface Messenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
        void tellCardWasSiezed();
        void askForAccount();
        void tellNotEnoughMoneyInAccount();
        void tellAmountDeposited();
        void tellBalance();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

BUT…we have grouped unrelated functionality together…yes, they are all messages, but they occur at different times during an ATM transaction.

```
public interface Messenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
        void tellCardWasSiezed();
        void askForAccount();
        void tellNotEnoughMoneyInAccount();
        void tellAmountDeposited();
        void tellBalance();
}
```

e.g. login Messages

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

Refactoring the Interfaces (1st pass)…

```
public interface Messenger {
        void tellCardWasSiezed();
        void askForAccount();
        void tellNotEnoughMoneyInAccount();
        void tellAmountDeposited();
        void tellBalance();
}
```

```
public interface LoginMessenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

Refactoring the Interfaces (2nd pass)…

```
public interface AccountMessenger {
        void askForAccount();
        void tellNotEnoughMoneyInAccount();
        void tellAmountDeposited();
        void tellBalance();
}
public interface ErrorMessenger {
        void tellCardWasSiezed();
}
public interface LoginMessenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

Refactoring the Interfaces (3rd pass)…

```
public interface AccountMessenger {
	void askForAccount();
	void tellBalance();
}
```

```
public interface DepositMessenger {
	void tellAmountDeposited();
}
```

```
public interface WithdrawalMessenger {
	void tellNotEnoughMoneyInAccount();
}
```

```
public interface LoginMessenger {
	void askForCard();
	void tellInvalidCard();
	void askForPin();
	void tellInvalidPin();
}
```

```
public interface ErrorMessenger {
	void tellCardWasSiezed();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

ISP – Example 1 – A Basic ATM

Using the Interfaces…

- Just choose which interface(s) are appropriate for the implementing class!
- Having refactored our interfaces, we are not forced to provide implementations for methods that we don't need.

```
public class Messenger implements LoginMessenger, WithdrawalMessenger
{
    //concrete implementations of the methods listed in the interfaces above.
}
```

```
public interface LoginMessenger {
        void askForCard();
        void tellInvalidCard();
        void askForPin();
        void tellInvalidPin();
}
```

```
public interface WithdrawalMessenger {
        void tellNotEnoughMoneyInAccount();
}
```

https://zeroturnaround.com/rebellabs/object-oriented-design-principles-and-the-5-ways-of-creating-solid-applications/

Example 2 -

An Advanced ATM

Interface Segregation Principle
If IRequireFood, I want to Eat(Food food) not,
LightCandelabra() or LayoutCutlery(CutleryLayout preferredLayout)

ISP – Example 2 – An Advanced ATM

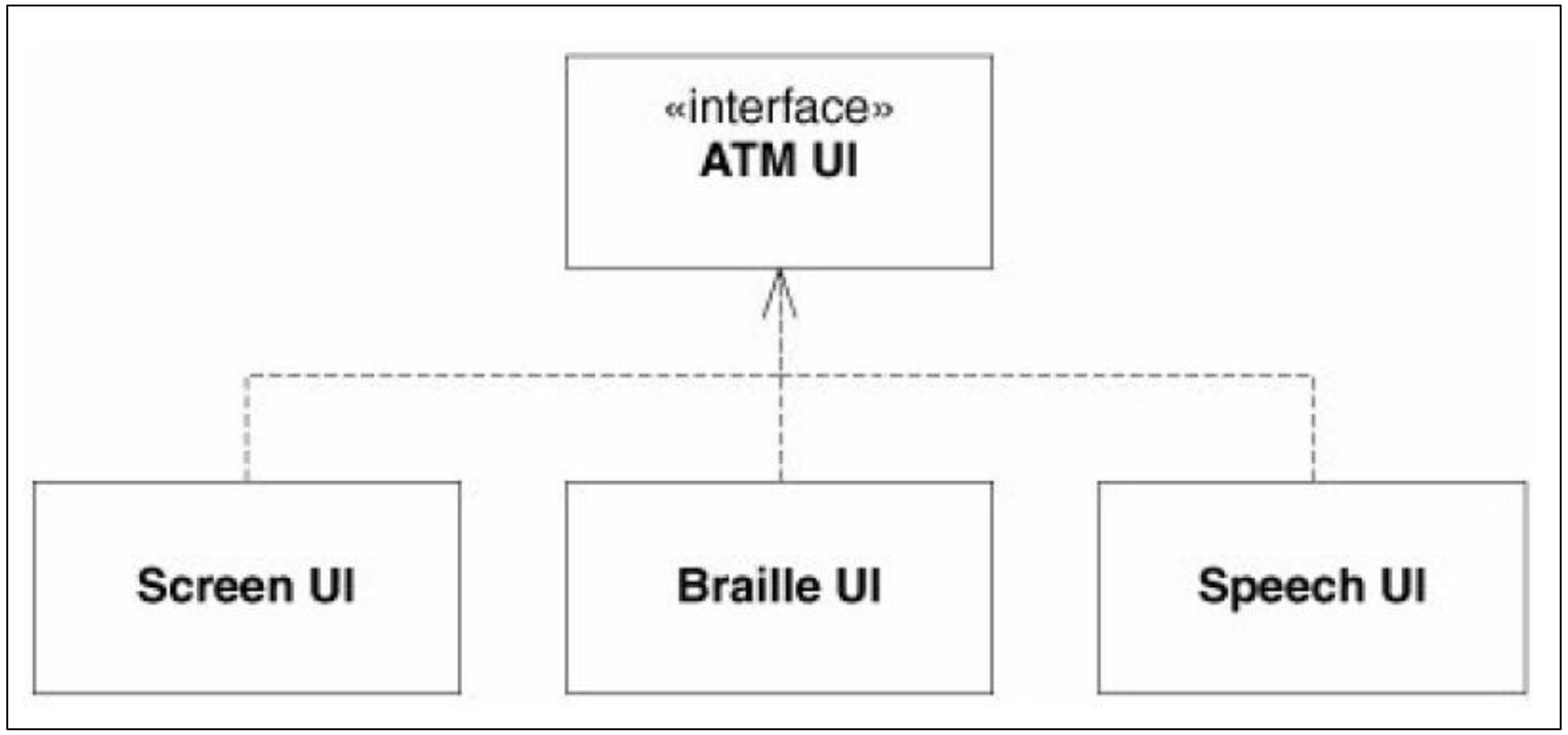
We now want ATM output to be:

- Printed on a variety of devices i.e. screen, braille tablet & speech synthesizer.

ISP – Example 2 – An Advanced ATM

We now want ATM output to be:

- Printed on a variety of devices i.e. screen, braille tablet & speech synthesizer.

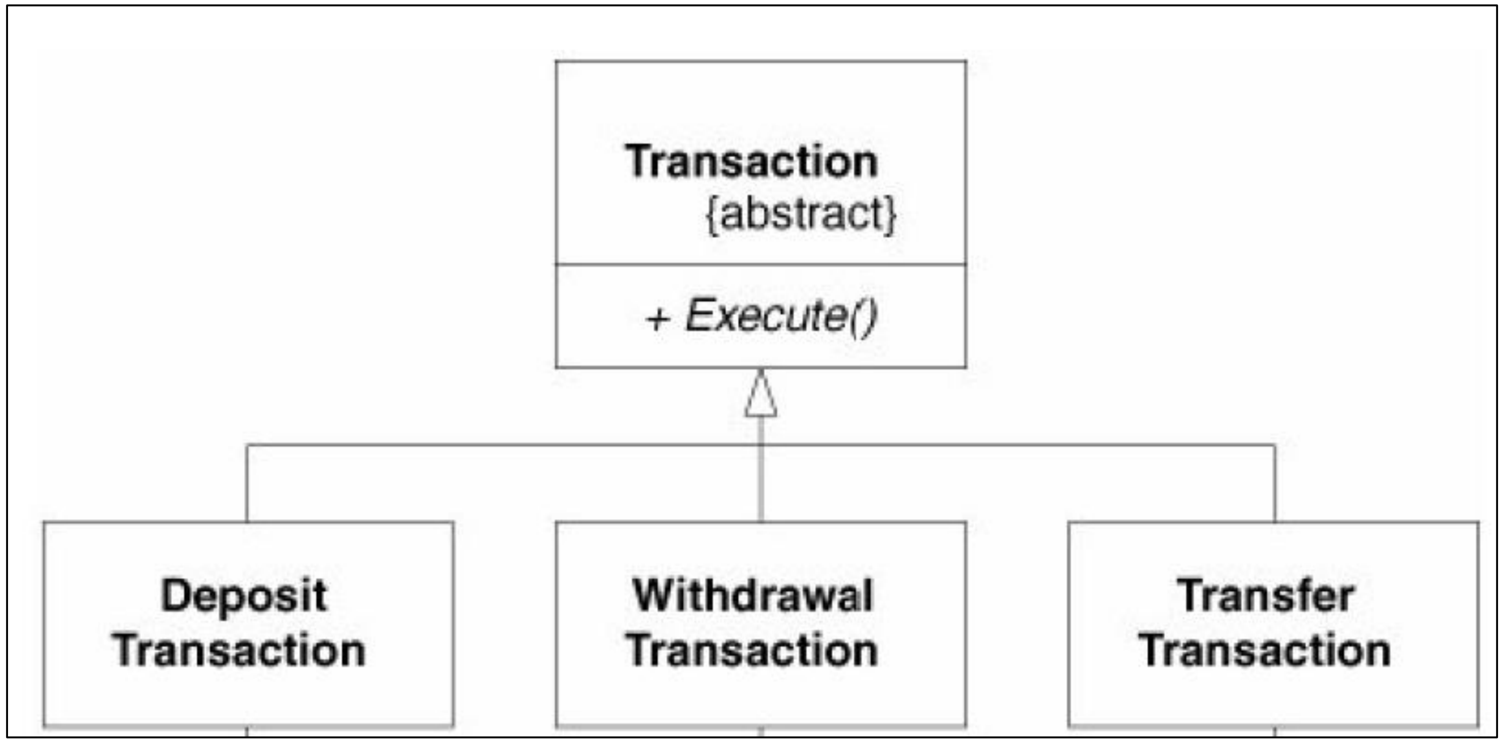


ISP – Example 2 – An Advanced ATM

Consider also that all ATM transactions are concrete implementations of the abstract Transaction class.

ISP – Example 2 – An Advanced ATM

Consider also that all ATM transactions are concrete implementations of the abstract Transaction class.

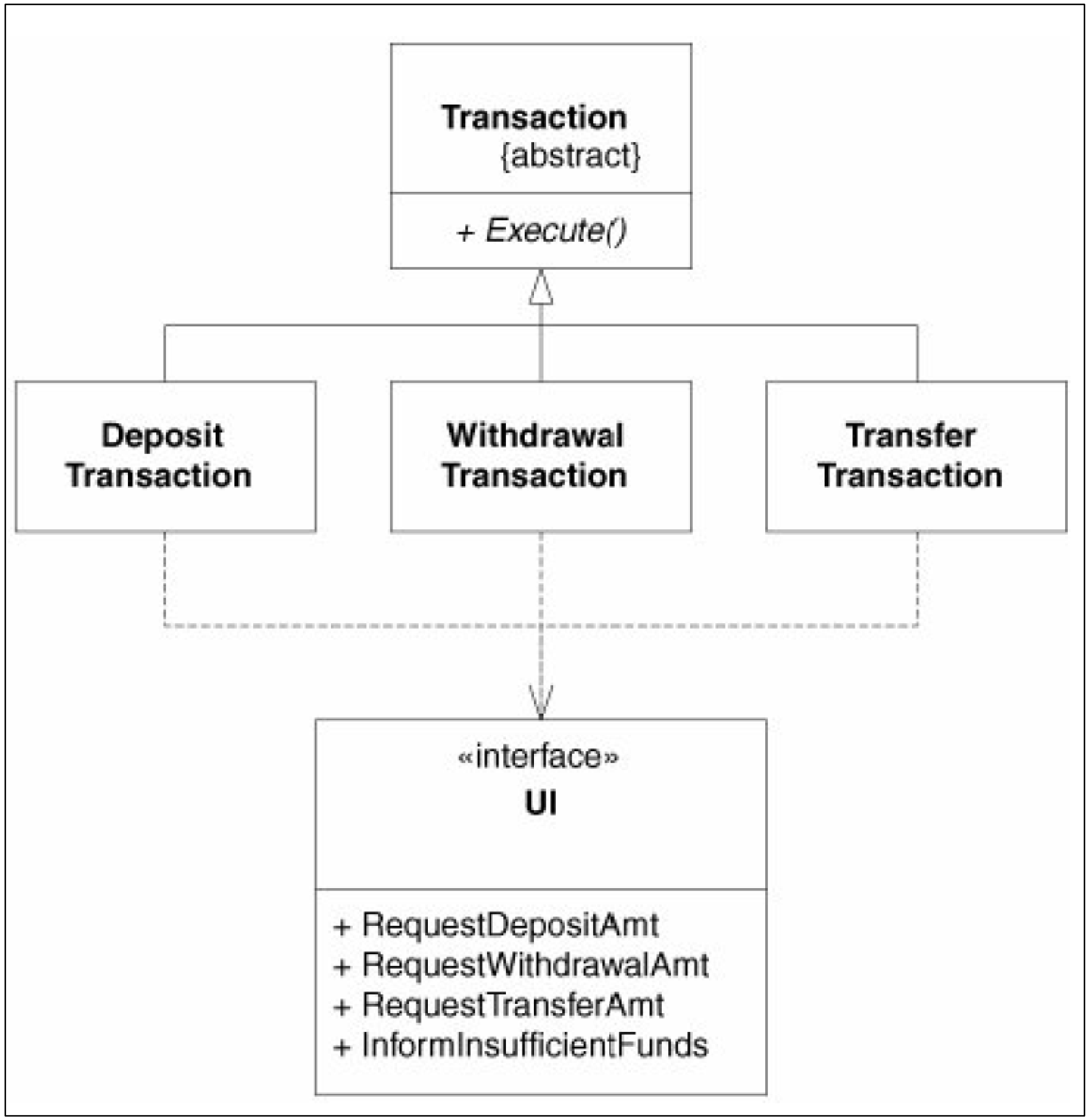


ISP – Example 2 – An Advanced ATM

Assume now that each concrete transaction class invokes UI methods.

For example:

- In order to ask the user to enter the amount to be deposited, the DepositTransaction object invokes the RequestDepositAmount method of the UI class.

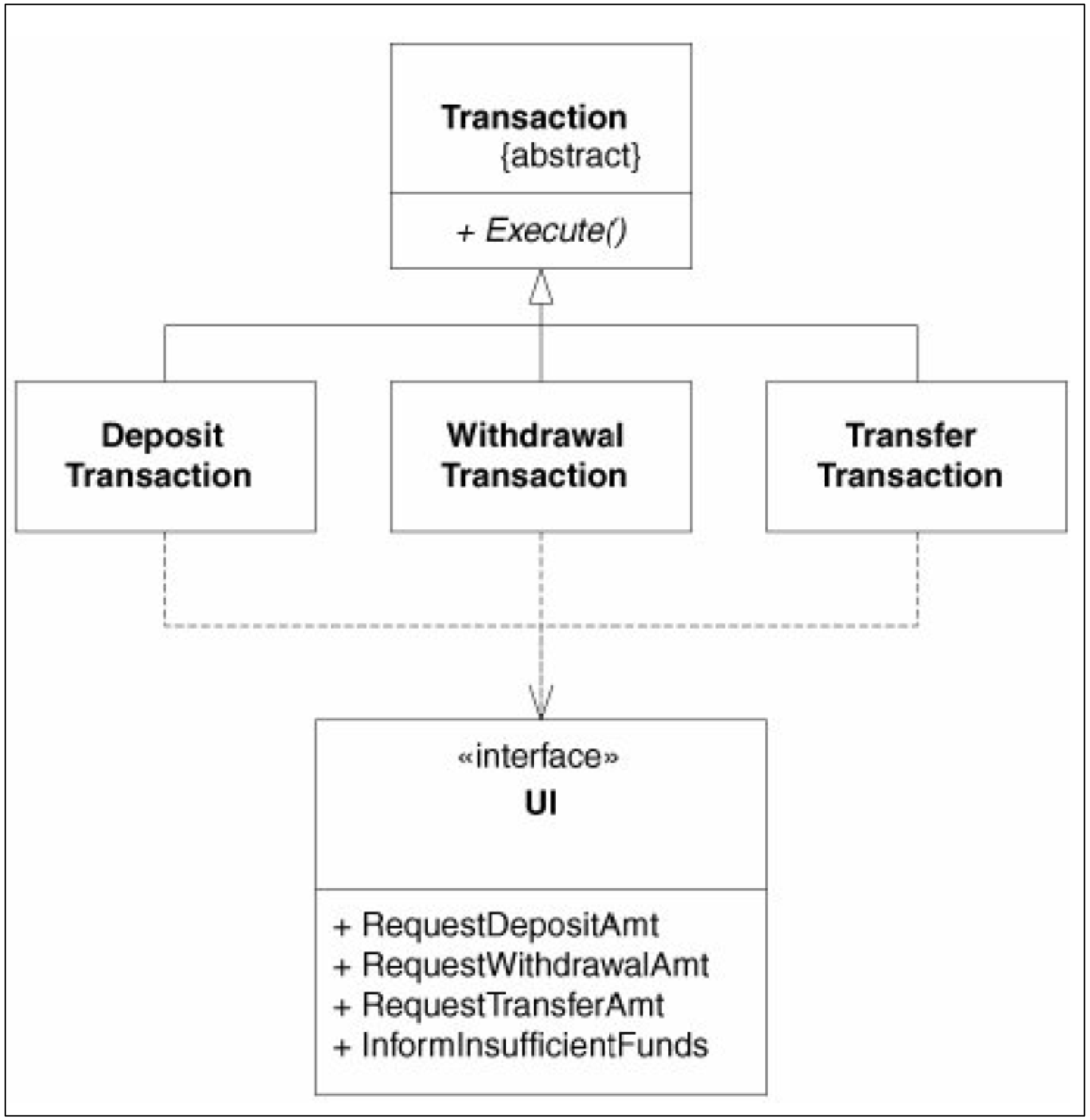


ISP – Example 2 – An Advanced ATM

Now consider that we need a new transaction, say, PayGasBillTransaction.

We would need to add new methods to the UI interface to deal with messages associated with the new transaction.

See any problems???

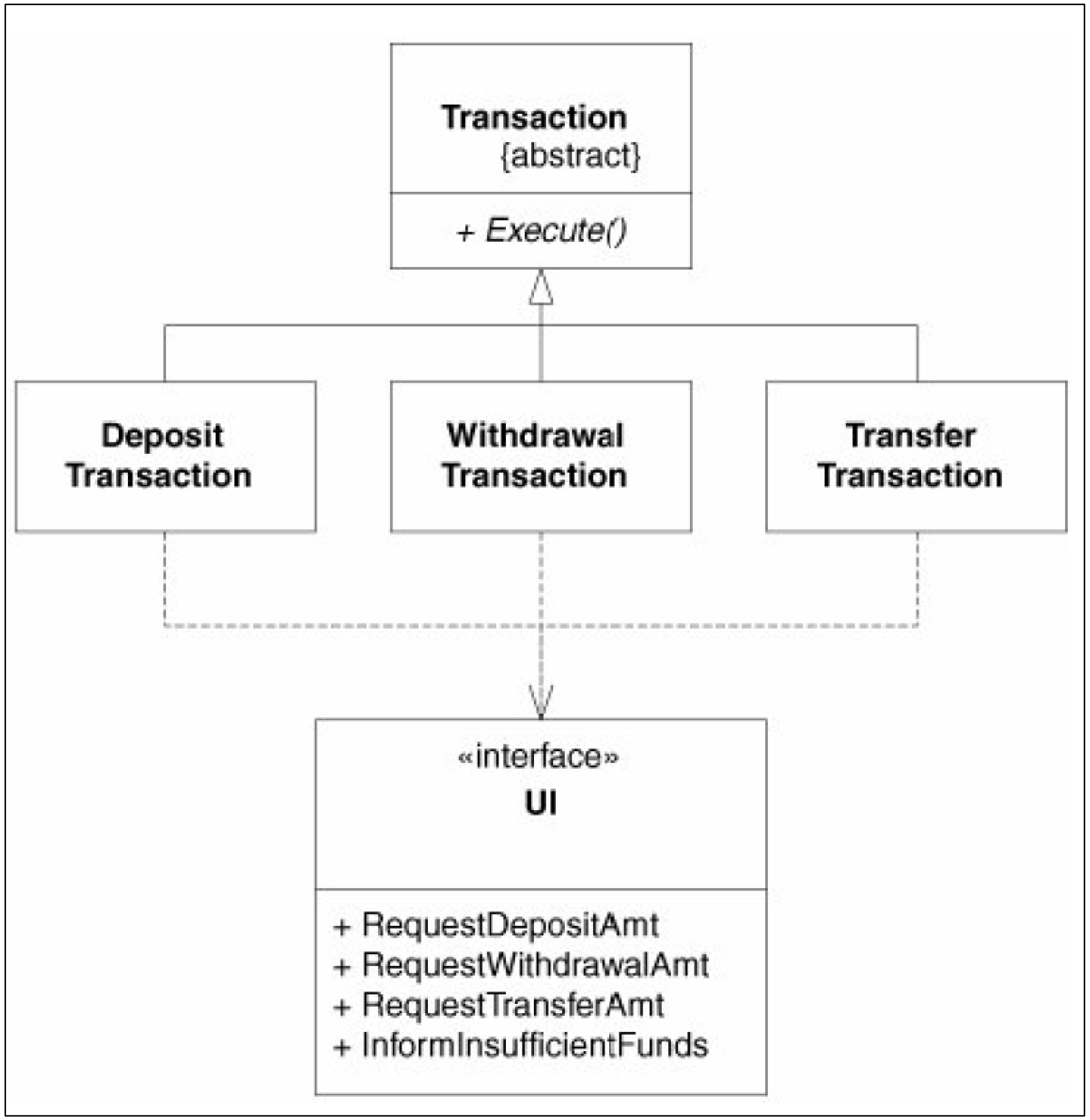


ISP – Example 2 – An Advanced ATM

Problem:

Since DepositTransaction, WithdrawalTransaction, and TransferTransaction all use the same UI interface, we are going to have to rebuild them, with code for the new methods.

If any of these transactions are components in other systems, we would have to redeploy, etc.

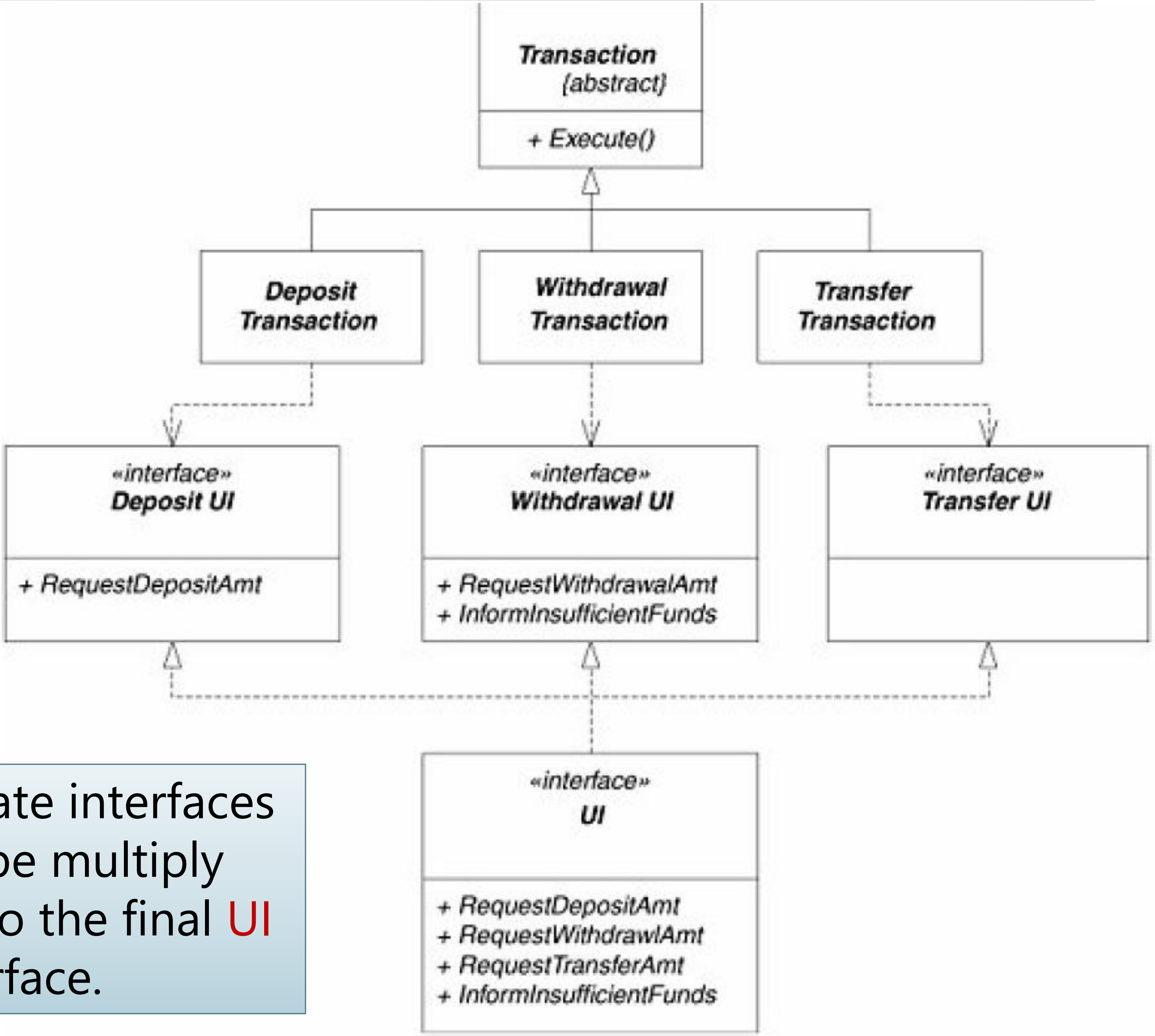


ISP – Example 2 – An Advanced ATM

Solution:

We can avoid this unfortunate scenario by segregating the UI interface into multiple interfaces.

These separate interfaces can then be multiply inherited into the final UI interface.



Interface Segregation Principle (ISP) - Recap

Clients should not be forced to depend upon interface members they do not use.

Interface Segregation Principle (ISP) - Recap

Clients should not be forced to depend upon interface members they do not use.

- Interfaces that are not cohesive are “fat”.
- ISP does not like "fat" interfaces.
- Break up “fat” interfaces into groups of cohesive methods.